Parallel and Distributed Computing (B4B36PDV)

Matěj Kafka, Michal Jakob

kafkamat@fel.cvut.cz

https://pdv.pages.fel.cvut.cz

Parallel and Distributed Computing

What is the difference?

Parallel computing

Utilize multiple computation units to get the result faster.

"single computer"
(shared memory)

Faster solution

Distributed computing

Utilize a network of separate computers to either get the result faster, or more reliably.

"multiple computers"
(message passing)

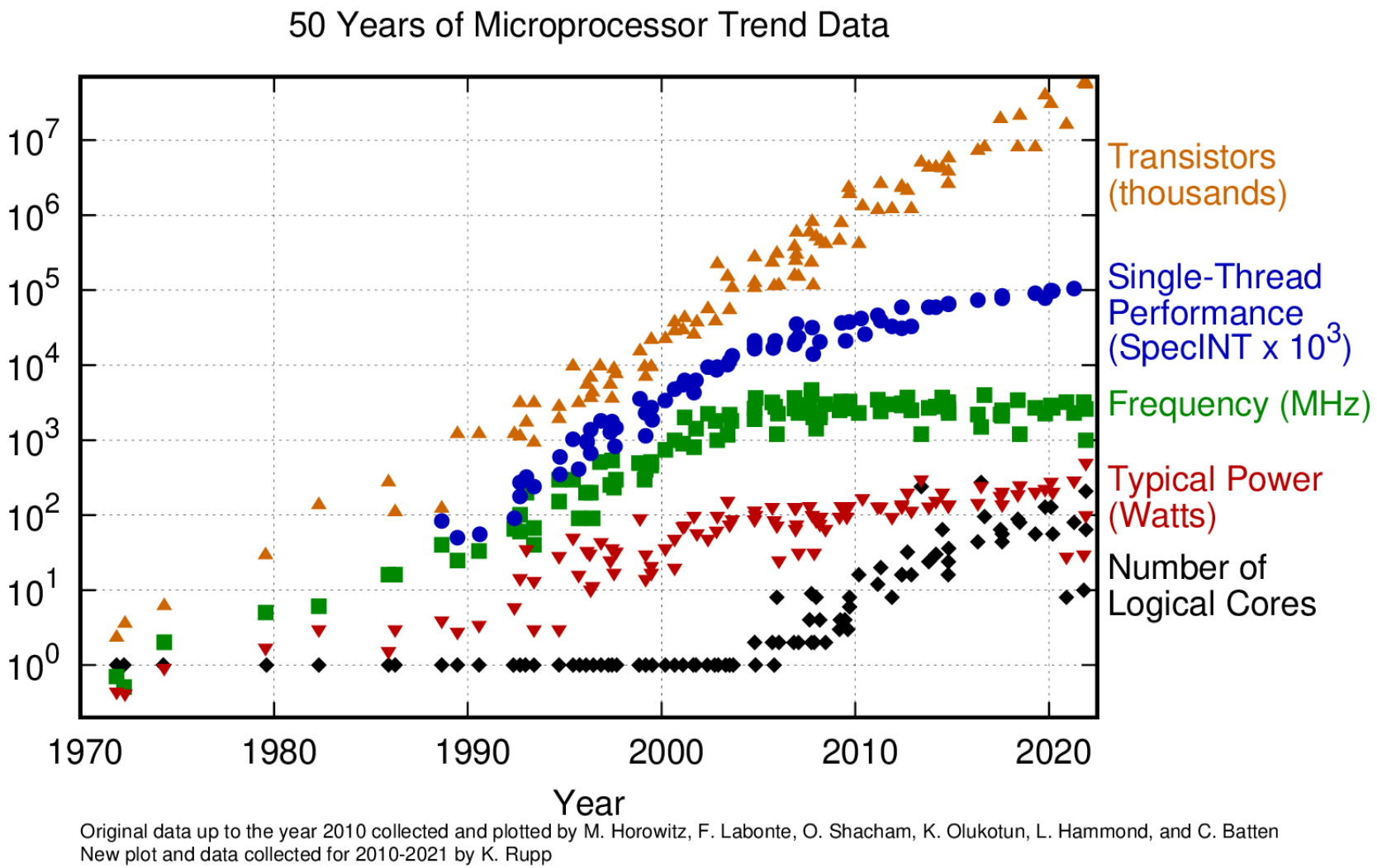
More robust system

Making programs run faster using parallelization

PARALLEL COMPUTING

Motivation

End of frequency scaling



Source: https://github.com/karlrupp/microprocessor-trend-data

Contemporary hardware

Threadripper PRO 7995WX

- 96 cores (192 hyperthreads)
- 12.6 TFLOPS

NVIDIA RTX 4090

- 16384 shader units
- 178.8 TFLOPS

Contemporary laptop hardware

Apple MacBook Pro (2024), 10 CPU cores, 1280 shader units

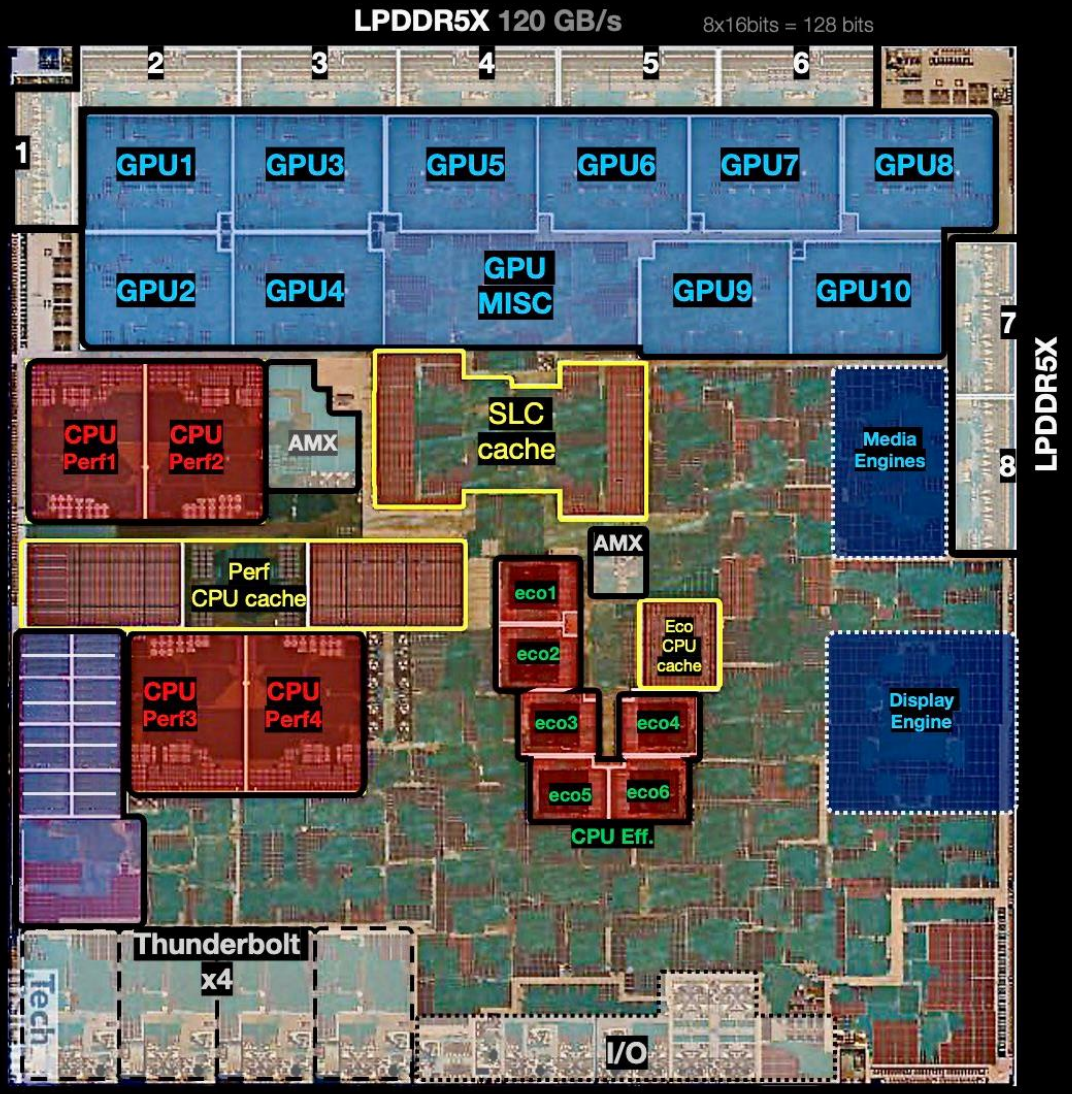
Apple Silicon

M4

28 billion transistors

N3E TSMC (2n gen. 3nm)

Die zise : 1.28mm x1,21mm



Frederic_Orange

Contemporary laptop hardware

Surface Laptop Studio 2, 14 CPU cores, 20 threads, 3072 shader units

Source: https://www.anandtech.com/

Compilers will not help us…

…yet

- For single-threaded programs, compilers work hard to make our programs fast (and tend to be good at it).
- Contemporary compilers will **not** magically make our programs multi-threaded.
- Libraries can often help, but we still need to know where and how we want to run things in parallel.
- Parallelism does not easily compose.

CPU-bound computation

DEMO 1: PARALLEL FOREACH

Brain-bound computation?

DEMO 2: STUDENT SUM

Amdahl's Law

Speedup is limited by serialized execution

$$S = \frac{1}{s + \frac{1 - s}{P}}$$

S = speedup

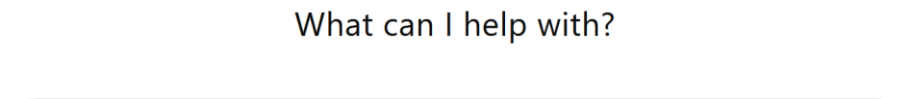
s = serial part

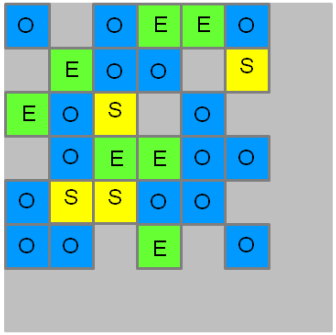
P = core count

https://www.desmos.com/calculator/7hday0d3ne

Will I ever use this in practice?

Why learn parallel computing?

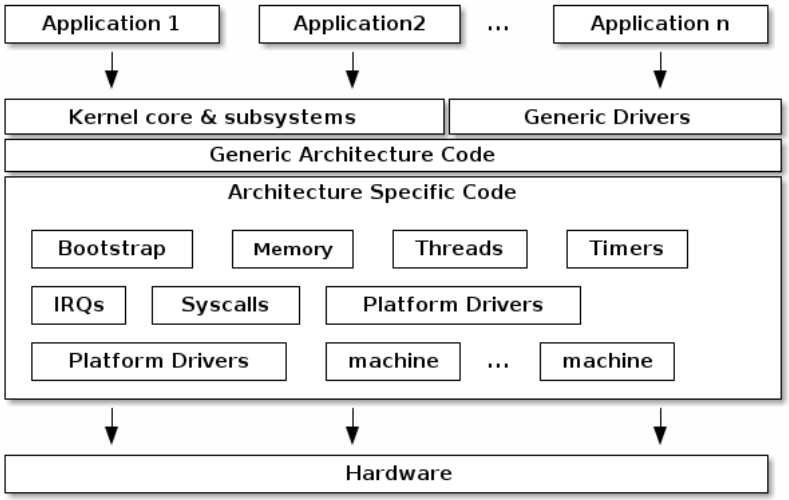




Eden Space

Survivor Space

Old Generation



B4B36PDV

ORGANIZATION

Who are we?

Lecturers

Matěj Kafka

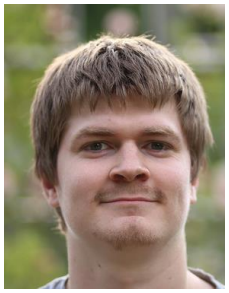
Michal Jakob

Tutors

Peter Macejko

Viktorie Valdmanová

Pavel Madaj



Max Hollmann

Jan Vanke

David Milec

What will we use?

Parallel computing

- C++20, OpenMP
- Linux / Mac / WSL
- JetBrains CLion / VS Code

- Knowledge from APO, OSY and ALG

Distributed computing

- Java 17
- Linux / Mac / Windows
- IntelliJ IDEA

- Knowledge from LGR and OSY

How do we evaluate?

- Assignments (50%)
  - 7 small assignments
  - 2 large assignments
- Implementation exam (15%)
- Theoretical exam (35%)

You need to achieve at least 50% from each part to pass.

How to succeed in PDV?

- **Review PRP, APO and OSY.** It will make the parallel part much easier.
  - C knowledge, pipelining, caches, threads, mutexes, race conditions, debugging
- Learn to combine high-level algorithmic **decomposition** with low-level **understanding of hardware**.
- **Think while debugging.** Randomly throwing code at the wall rarely fixes multithreading issues.
- Use "AI" chatbots wisely.

QUICK REVISION OF CPU ARCHITECTURE

Why is CPU architecture relevant?

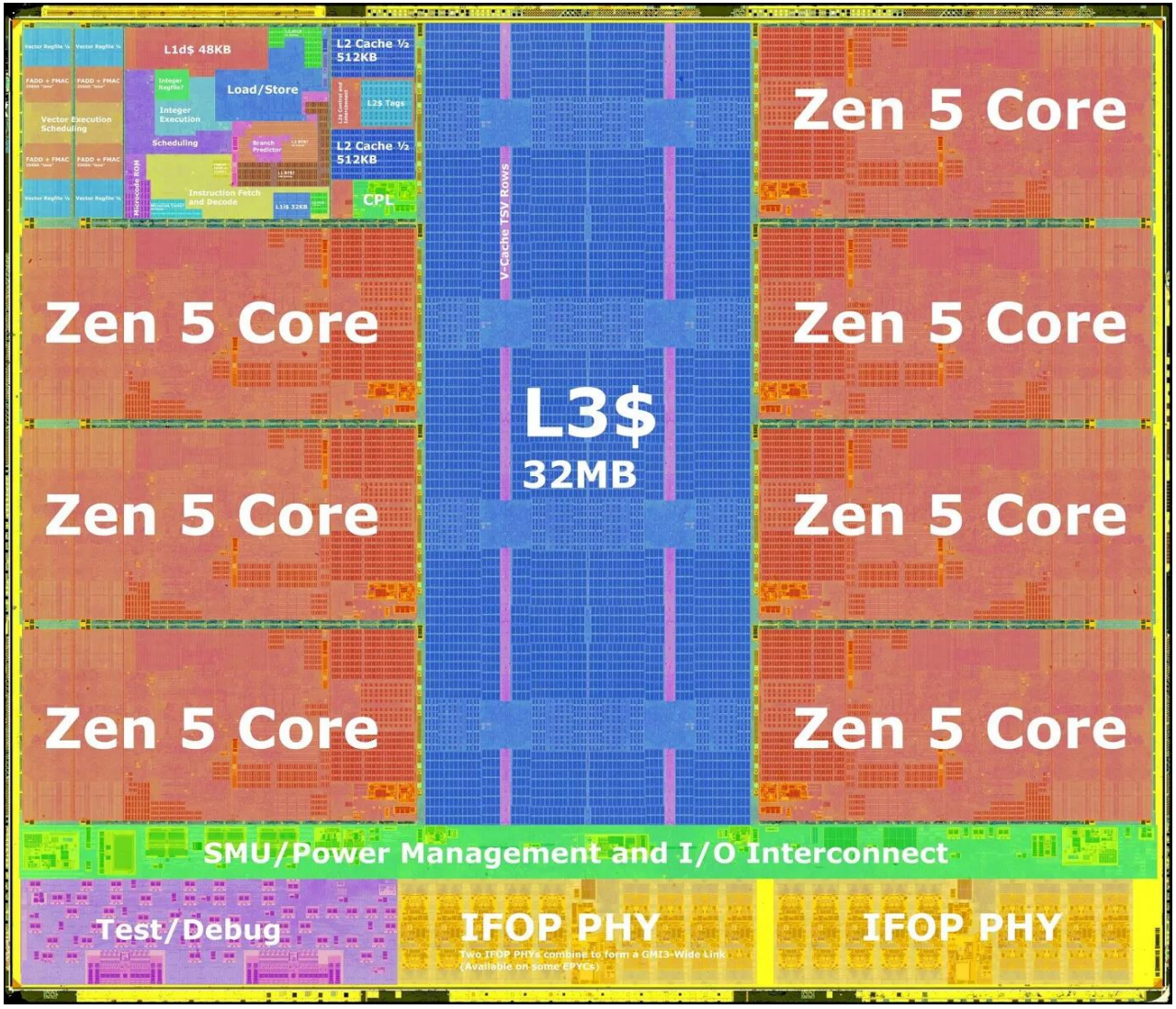
Matrix-vector multiplication

```
float x[SIZE];
float y[SIZE];
float A[SIZE * SIZE];

// VERSION 1
for (size_t i = 0; i < SIZE; i++)
    for (size_t j = 0; j < SIZE; j++)
        y[i] += A[i * SIZE + j] * x[j];

// VERSION 2
for (size_t j = 0; j < SIZE; j++)
    for (size_t i = 0; i < SIZE; i++)
        y[i] += A[i * SIZE + j] * x[j];
```

Which version will be faster?

CPU structure

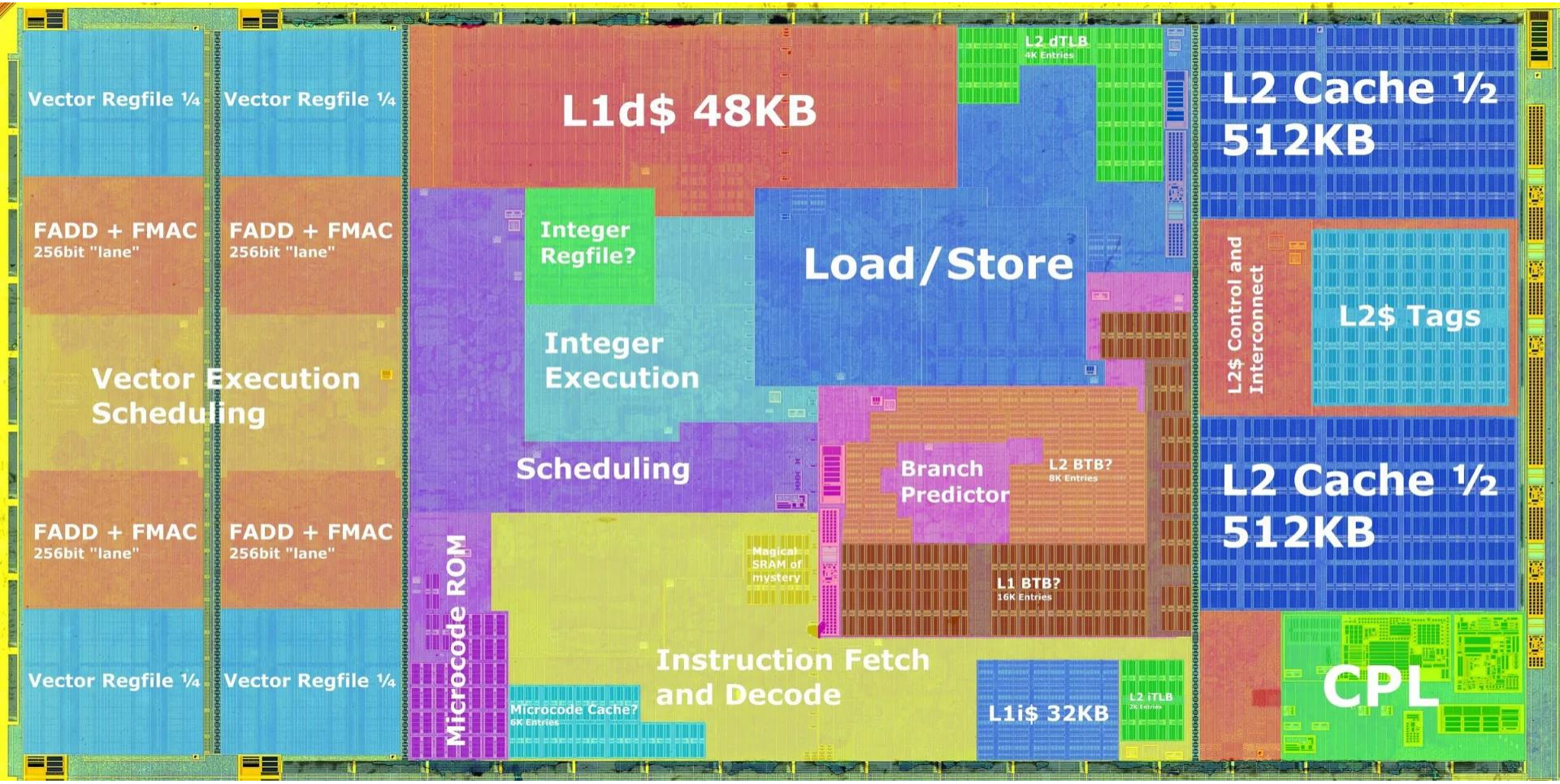
AMD Zen 5 annotated die shot



Source: https://nemez.net/die/

CPU structure

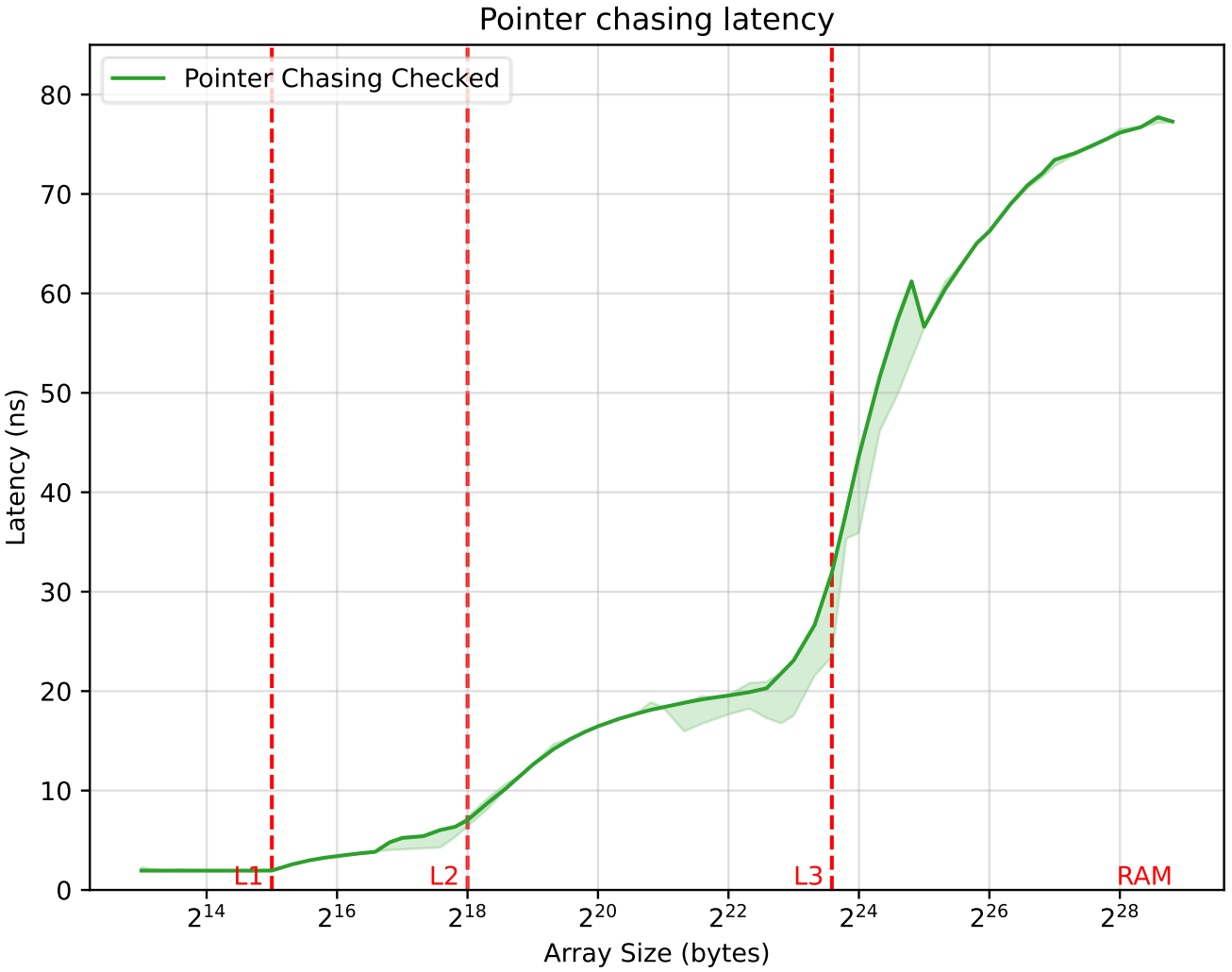
AMD Zen 5 annotated die shot



Source: https://nemez.net/die/

CPU cache latency

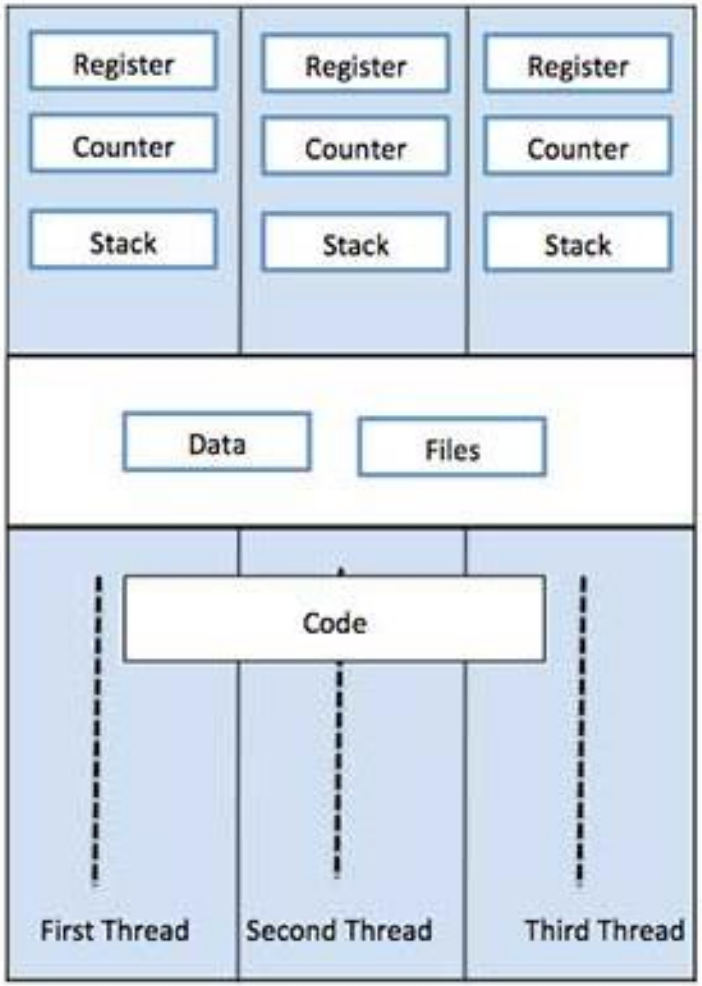
Intel i7-10750H



Source: https://curiouscoding.nl/posts/cpu-benchmarks/

Hardware threads (MIMD)

Multiple instructions, multiple data



Single Process P with three threads

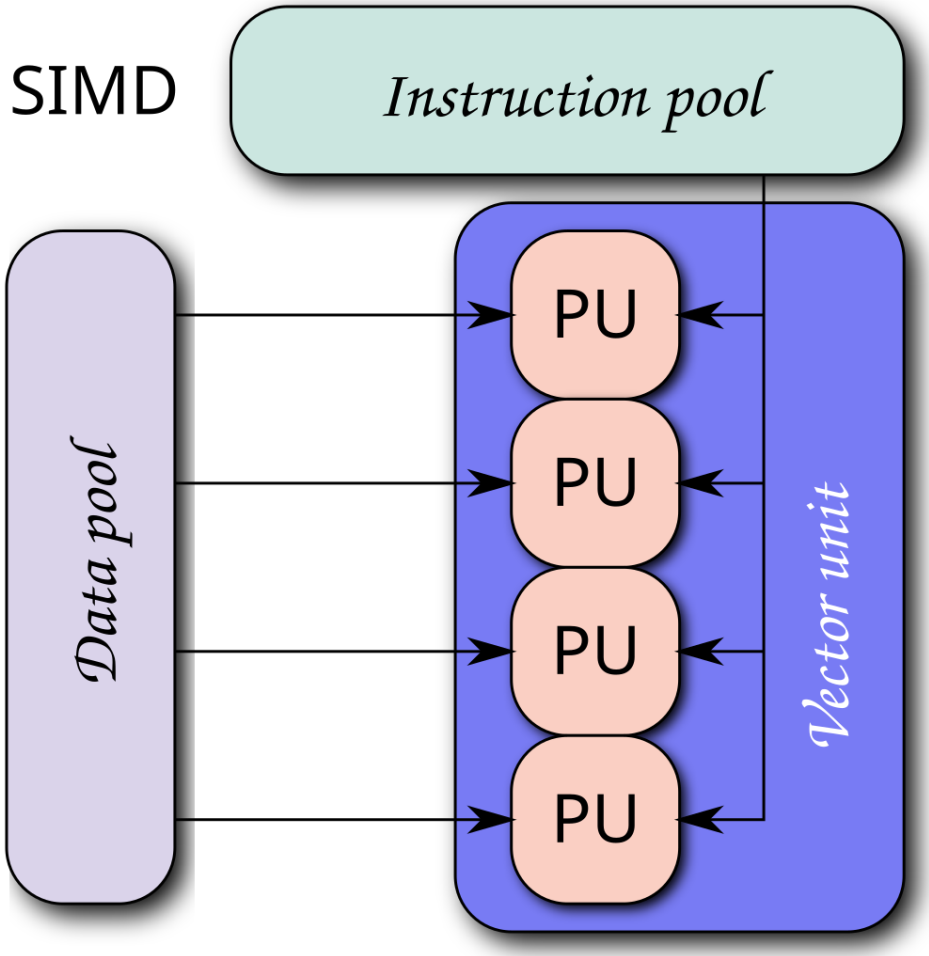
Multiple threads of execution, each one with separate control unit and data

Multi-core CPUs (you should already know from OSY), hyper-threading

Source: https://www.tutorialspoint.com/operating_system/os_multi_threading.htm

SIMD

Single instruction, multiple data



Single pipeline, single control unit, multiple ALUs

"data parallelism"

GPUs, vector ALUs in CPUs, various parallel accelerators

Source: https://commons.wikimedia.org/wiki/File:SIMD2.svg

Why is CPU architecture relevant?

Array sum ("reduction")

```
float array[SIZE];
float sum = 0.0f;

// split parts of the for loop
// between multiple threads
# pragma omp parallel for
for (size_t i = 0; i < SIZE; i++) {
    sum += array[i];
}
```

Slow, sum is shared between all threads! ("true sharing")

Why is CPU architecture relevant?

Array sum ("reduction"), improved version?

```
float array[SIZE];
float sums[THREAD_COUNT] = {0.0f};

// split parts of the for loop
// between multiple threads
# pragma omp parallel for
for (size_t i = 0; i < SIZE; i++) {
    sums[THREAD_ID] += array[i];
}
```

Slow, sums are in the same cache line! ("false sharing")

Resources

Interesting articles used in this lecture (not mandatory)

- https://www.cs.cmu.edu/~15418/schedule.html
  course from Carnegie Mellon University, great slides, similar area but more in-depth
- https://www.youtube.com/watch?v=eanvgGt-D1o
  old recording of the first lecture from the course above
- https://curiouscoding.nl/posts/cpu-benchmarks/
  very well-done benchmarks of CPU cache latency
- http://gotw.ca/publications/concurrency-ddj.htm
  "The Free Lunch Is Over" by Herb Sutter – article explaining why parallelism is now the answer to improving performance